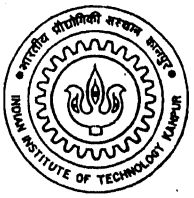
Applying Parametric Query Optimization to Non-Parametric Query Metrics

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

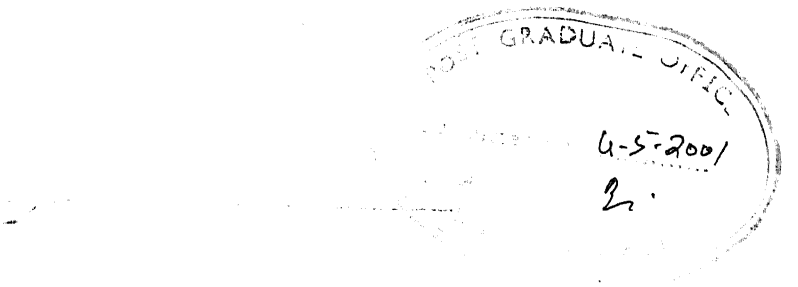
V R L Swamy Vadali



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

May, 2001



Certificate

This is to certify that the work contained in the thesis entitled "*Applying PQO to Non-Parametric Query Metrics*", by *V.R.L.Swamy Vadali*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

May, 2001

Ganguly

(Dr. Sumit Ganguly)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Abstract

Query optimization in database systems having non-parametric query metrics cannot be done using dynamic programming approaches. In this thesis, we propose an approach *to solve the query optimization problem of complex non-parametric query metric*

$$B(p) = max(a_0(p), a_1(p), \ldots, a_n(p))$$

by formulating *Affine problems* from it. This approach can be applied to other complex non parametric query metrics which are monotonous.

Acknowledgments

I take this opportunity to express my sincere and whole-hearted gratitude towards my thesis supervisor Dr. Sumit Ganguly for his support and guidance throughout my thesis. His dedication and devotion towards research has always been a motivation for me. His suggestions and criticisms has helped me a lot to enhance my understanding of the subject. His patience and constant encouragement always inspired me to tackle every obstacle boldly.

I am also thankful to all the faculty members of Department of Computer Science and Engineering, for the invaluable knowledge they had imparted to me. I would also like to thank the technical staff of the department for maintaining excellent and round the clock computing facilities.

I am also grateful to all my batchmates of great mtech99 who made my stay at IIT kanpur the most memorable and comfortable one. I would never forget the consecutive night-outs we had in thesis room at endsems and project deadlines. I would also express my thanks to Hrishi and Vasudev for helping me initially to pick up things quickly. I am also very grateful to Atul Kumar and Santosh Singh for helping me out during my stay at IIT kanpur.

I would like to express a very special thanks to Amma and Näana and sisters Akka and Chelli, for the love and encouragement that helped me to come up so far. Above all I would like to express my gratitude to God for giving me enough opportunities and courage.

V.R.L.Swamy Vadali

Contents

List of Figures

Chapter 1

Introduction

1.1 Query Optimization

With the rapid strides in the computing and information technologies resulting in need for managing voluminous data, Database Management Systems have become a *de facto* standard for data processing and information retrieval systems. The Database Management Systems offers all the features for efficient, robust, fault-tolerant and user amenable data access by abstracting data as Objects, Relations or other hierarchal concepts. The Internet revolution and data-intensive applications like GIS, multimedia repositories has triggered the need for lowering the response time of users request to retrieve data.

A request for data retrieval to a Database System is specified as an non-procedural SQL query. Originally. SQL was called SEQUEL (for Structured English QUEry Language) and was designed and implemented at IBM Research as the interface for an experimental relational database system called SYSTEM R. SQL is now the de facto standard Relational Database Language. A query in SQL specifies the relations to be accessed, the attributes to be listed and the predicates which are to be satisfied by every tuple.

For example, in Structured Query Language we can express the query to retrieve the names of all the students of CSE department from the relation STUDENT(NAME,ROLL,DEPT) as

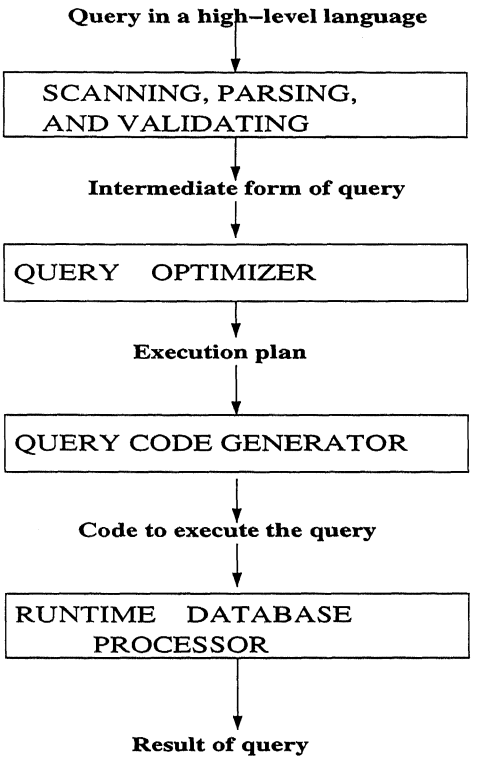


Figure 1.1: Steps of processing a query in high-level language.

```
Select Name
From STUDENT
Where DEPT = "CSE"
```

The SQL query specifies only the predicates which must be satisfied by tuples selected from the relations given. The SQL query keeps away the burden of specifying the *access path* (primary or secondary) to be used, the *join methods* (Nested loop, Sort-Merge or Hash join) or the *join order* to be applied for processing the query. The overall strategy of processing the query given in non-procedural languages like SQL or embedded in high level languages is shown in the Figure 1.1.

A query expressed in a high level query language must be first scanned, parsed and validated [NAEL94]. The *scanner* identifies the language tokens in the query and the *parser* checks whether the query is conforming to the syntax rules of the query language. The query is then *validated*, by checking whether all the relations are existent or attributes are valid and expression is semantically meaningful. The query is then represented internally, usually as a tree or a graph data structure, which is also called a *query tree* or *query graph*. Now an *execution strategy* for the query graph must be established by the database specifying the access path to be selected, the join technique and the join order.

The data about the data in the database, *metadata* is stored in the System Catalogs. The system catalogs store the *schemas* or descriptions of the database that the DBMS maintains. It describes the conceptual database schema, the internal schema, any external schemas and the mapping between the schemas at different levels. From the catalogs the information regarding the access paths available, size of relations, selectivity of predicates can be derived. The information in the catalogs are periodically updated by some module of the backend database engine.

A query usually has multiple execution strategies and the technique of choosing the best one with respect to execution cost from feasible set of plans is termed as *query optimization* [KS86]. The *query optimizer* module produces an execution plan, and the *code generator* generated the code to execute that plan. The *runtime database processor* generated the query result by running the query code, whether compiled or interpreted.

The above setup provides for a large room of options to the *query optimizer* to choose an execution plan. In case of joins or where the cardinality of the relations is very large, the cardinality of intermediate relations can vary substantially, hence the execution and response times. In data intensive applications or in real time informations systems, where response time is at premium, applying optimization techniques to query graph can substantially improve the performance of the system.

All query optimization techniques can be categorized either as *cost-based optimizers* or *rule-based optimizers*. The cost based optimizers base their decisions on some cost models.

1.2 Cost based and Rule based optimizers

The *Rule based optimizers* use a rigid set rules to determine an execution plan for any SQL statement. The rule description languages are used to specify the general algebraic transformations or execution strategy for a given type of query. The current state of the database has no effect on the execution plans. The decisions are taken statically, a given query would always be transformed to the same query execution plan.

The *cost based optimizers* base their execution plan selection decision on some cost models and choose the plan among the feasible plans having the minimum cost. The cost of executing a query can be modeled in terms of accesses to secondary storage, or response time to fetch the request data etc. The cost of executing a query includes the following components-

- Access cost to secondary storage.
- Storage cost.
- Computation costs.
- Communication costs.

In large databases, minimizing the access cost to secondary storage is needed. While in smaller databases, where most of the data involved can be kept in main memory the main concern is minimizing the response time. On the other hand in case of distributed databases where multiple sites are involved, the communication costs also have a substantial impact. The current state of the database is obtained from the catalogs and execution plan having the minimum cost as per present state of database is chosen.

1.3 Parametric Query Optimization

Cost-based optimizers optimize queries based on the some cost models. The cost of a query execution plan depends on many parameters like available memory, cardinality

of base relations, size of intermediate results, selectivities of predicates, available access paths, processor speeds, communication links, disk latencies etc. The query optimizers compile a query into best execution plans assuming that all the parameter values are known at compile time. Practically the parameter values change between the compile time and runtime, having substantial impact on costs. Also in case of embedded SQL constructs, the predicates may have unspecified variables which are known only at the run time. Since the execution environment and database system is constantly changing, statically optimized query plan at compile time might turn out to be suboptimal at runtime. This result was shown by Graefe and Ward[GrW89]. A typical scenario of this would be a web based status enquiry in Railway reservation system. The user may try to enquire the current status of a ticket (PNRNO = x).

```
Select *
From RAILRES
Where PNRNO = x
```

Depending on the value of the variable x, the optimal plan may choose one of the access path from either sequential scan or index on PNRNO.

The problem of parametric query optimization is computing the different optimal plans considering the cost effecting parameters. Typically a single plan would not be optimal for all values of the parameters. A plan would be optimal only for a subset of values called region of optimality of the plan. One naive approach would be to find the plan optimal for each value of parameter, but this would be computationally exhaustive. *The parametric query optimization problem tries to compile a query in to a set of plans called parametric optimal set of plans, each optimal for some range of parameter values.*

1.4 Non-Geometric and Geometric Approaches

Various non-geometric approaches for this problem have been designed by Ionnidis etal., [INS+92], Cole and Graefe [CG94], Antoshenkov [Ant93], Sumit Ganguly

and Krishnamurthy [GK94], and Sumit Ganguly [Ganguly98]. [INS+92] follows a randomized approach. The Cole and Graefe [CG94] technique is based on partial ordering of costs of different plans. These techniques also generate a number of suboptimal plans. And above all these non-geometric techniques does not find the range of parameter values where a plan is optimal.

The geometric approach for parametric query optimization problem called *Iso-point* method was introduced in [Ganguly98]. The algorithm for Linear one parameter cost metric is in [Anjali99]. The algorithms discussed in the paper Ganguly98 has been extended to ternary linear cost functions and binary non-linear cost functions in [Prasad99]. The geometric approaches for n parameter linear and non-linear cost functions have been successively solved in [SGPrUmAn2001]. The problem of parametric query optimization for linear and non-linear parameterized cost equations has been solved. [SGPrUmAn2001] gives a $O(\nu+\eta)$ algorithm for the linear parametric query optimization problem.

Some non-geometric approaches for parametric query optimization for two different classes of query graphs, namely linear and star query graphs are in [SVUMRao99].

1.5 PQO problem

In this section let us define the Parametric Query optimization problem. The definition is same as given in [Ganguly98].

Let s_1,s_2,...,s_n denote n parameters, where each s_i quantifies some cost parameter, such as selectivity, table sizes, available memory etc. In order to take into account the possible variation of these parameters, a logical possibility is to compile a query into a set of optimal plans called the parametric optimal set of plans. For every legal value of the parameters s_1,...,s_n, there is a plan in the parametric optimal set that is optimal for that value and vice-versa. The region of optimality for a plan p is defined by the set-

$$\{(s_1, \ldots, s_n \mid \ p \textit{ is optimal at } (s_1, \ldots, s_n)\}$$

The problem of parametric query optimization is to find the parametric optimal set

of plans and the region of optimality for each parametric optimal plan.

The parametrized cost equation can be affine or include nonlinear terms, and number of parameters can be n. The PQO problem had been solved for both the linear and nonlinear case as well.

1.5.1 Affine problems

This section gives the definition of *Affine problems* and some properties of it. These have been discussed in [SGPrUmAn2001].

Definition 1: A *PQO* problem is said to be **affine** if the following three conditions hold.

1. The feasible space is a convex polyhedron of R^n.
2. For $\mathbf{x} = (x_1, x_2, \ldots, x_m) \in$ F, the cost function C(p, $\mathbf{x}$) is an affine function of $\mathbf{x}$, that is, C(p, $\mathbf{x}$) has the form

$$C(p, \mathbf{x}) = a_1(p) + a_2(p)x_1 + a_3(p)x_2 + \ldots + a_{n+1}(p)x_n$$

3. There exists a function optimize($\mathbf{x}$) that returns the set of plans that are (equally) optimal at a point $\mathbf{x} \in$ F.

Conditions 1 and 2 in Definition 1 are "structural" properties. The third condition is about the existence of a computable (and hopefully easy) procedure to solve the non-parametric problem.

We now discuss some simple and useful properties of affine *PQO* problems.

1.5.2 Properties of affine *PQO* problems

In this section, we state some properties of affine *PQO* problems. These properties have been quoted from [SGPrUmAn2001].

- *Property 1. Convexity of Regions of Optimality for Affine* PQO .

  This property states that the region of optimality of a plan p, R(p) is a convex polyhedron. The convexity theorem is given in [Ganguly98].

- *Property 2. An n-parameter affine cost function restricted to a k dimensional hyperplane in F is affine in k parameters.*

1.5.3 Examples: Affine problems

Here we discuss some examples of some query optimization problems which are Affine in nature. More information about these is in [SGPrUmAn2001].

- **Single unknown selectivity**

  Let QR be a query having a single predicate, and the selectivity of predicate is unknown. Assuming uniform distribution of data, using the cost models([SAC+79]), the parameterized cost equation of QR can be expressed as $C(p, s) = a_0(p) + a_2(p) \cdot s$. Let s be normalized such that $0 \leq s \leq 1$. The cost is linearly dependent on s and the problem can be expressed as affine PQO.

```
QR:     Select *
        From  STUDENT
        where HALL = 4
```

- **Load Balancing in Distributed Databases**

  In a distributed query processing environment, let a_i(p) denote the resource consumption by plan p at site or *processori*, having n processors or sites. With each site(or processor) a load factor l_i is attributed. The effective resource consumption can be modeled as

  $$C(p, l_1, l_2, \ldots, l_n) = a_1 \cdot l_1 + a_2 \cdot l_2 + \ldots + a_n \cdot l_n . 1 \leq l_i \leq \infty, 1 \leq i \leq n.$$

  This can be expressed a a union of n Affine PQO problems.

- **Optimal Affine Approximation for Parallel and Distributed Databases**

  In a distributed query processing environment let a_i(p) denote the resource consumption by the plan p at site i or processor i as the case may be. The cost metric is:-

  $$B(p) = max(a_0(p), a_1(p), \ldots, a_n(p))$$

  These cannot be solved by applying dynamic programming approaches and are difficult to optimize.

1.6 Motivation

The current computing trends, made it necessary to avail the techniques for considerable reductions in the response time of query execution along with fault-tolerance. The parallel and Distributed databases are gradually establishing as the best alternative for data-intensive and real time applications. In parallel and distributed databases, the cost models applicable to query execution plans are highly complex. They are not parameterized and do not satisfy the principle of optimality required for applying dynamic programming methodologies. A nice approach would be to extend Parametric query optimization techniques for optimizing complex query metrics.

1.7 Our Contribution

In this thesis we design algorithms to solve the problem of optimizing the complex query metrics using PQO techniques. The overall approach is geometric one. The query cost metric used is:

$$\mathbf{B}(p) = \mathbf{max}(a_0(p), a_1(p), \ldots, a_n(p)).$$

We have also extended our algorithm for n parameters query metric case.

1.8 Organization of the Thesis

Chapter 2 gives the detailed explanation of the algorithm designed by us for three parameter metric. First the properties of the cost coordinate space along with parameter space are discussed.

Chapter 3 extends the algorithm to n parameter complex metric.

Chapter 4 discusses some conclusions we arrived at and looks at future research work.

Chapter 2

Algorithm for 3 parameters

This chapter discusses the basic idea of how to solve the optimization problem for complex query metrics. Optimization of queries in distributed and parallel environment yields complex non-parametric cost metrics. These complex query metrics are computationally intensive and cannot be solved using dynamic programming approaches. An approach for solving these is to use Parametric Query Optimization (PQO) which gives us *Best Parametric approximates*. We shall discuss our algorithm for an example complex query metric.

2.1 Example cost metric

In a distributed query processing environment let a_i(p) denote the resource consumption by the plan p at site i or processor i as the case may be. The cost metric is:-

$$B(p) = max(a_0(p), a_1(p), \ldots, a_n(p))$$

The above cost metric is difficult to optimize since it is not parametric and solution techniques like dynamic programming are inadequate. So one intelligent approach is to apply PQO techniques by framing an appropriate Parametric Query optimization problem, which would give us the best approximate.

2.2 Overview of Approach

The PQO approach for best approximate of conventional non-parametric query optimization problems has the following two logical steps:

- *Formulating a problem as a best affine approximation*
- *Computing the best affine approximation*

2.2.1 Formulating a problem as a best affine approximation

Consider the query metric for the distributed query processing environment. The cost metric is defined by B(p) = $max_{i=0}^{n} a_i(\mathrm{p})$, where $a_i(\mathrm{p})$ denotes the resource consumption at some site i. We first formulate the PQO problem, represented by PQODD.

$$PQODD : C(p, \mathbf{x}) = \Sigma a_i(p) x_i,\ 1 \leq x_i < \infty,\ 0 \leq i \leq n.$$

Here x_0,x_1,...,x_n are the artificial parameters. The PQODD problem is not affine and so we construct n+1 affine PQO problems out of it as follows.

$$PQODD^i : C(p, \mathbf{s}) = a_i(p) + \Sigma_{j=0}^{n, j \neq i} a_j(p) \cdot s_j,\ s_j = \frac{x_j}{x_i};\ 0 \leq s_i \leq 1$$

Now find the parametric optimal plan in the parameter space which is having the minimum value of B(p) metric. This is the *best affine approximation* of the plan that is optimal with respect to the metric B. The best approximate can be efficiently computed without generating the *Parametric Optimal Set (POS)* of plans.

Consider the case when B(p) = max(a_0(p),a_1(p)). The following are the two affine PQO problems:

$$PQODD_0 : C^{(0)}(p, x) = a_0(p) + a_1(p) \cdot x\ 0 \leq x \leq 1$$

$$PQODD_1 : C^{(1)}(p, x) = a_1(p) + a_0(p) \cdot x\ 0 \leq x \leq 1$$

In the next section we will discuss the algorithm for best affine approximation which has been designed by *Sumit Ganguly* [SGPrUmAn2001].

2.2.2 Computing the best affine approximation

We have to find the parametric optimal plan having the minimum value of B metric. The *EQUILINE* refers to the line $a_0(p) = a_1(p) = \ldots = a_n(p)$ in the n dimensional cost coordinate space. The basic idea is, first we check whether the *EQUILINE* cuts the convex hull CH or not. We optimize at x=0 and x=1.

If $(a_1(p) \geq a_2(p) \mid p = \text{optimize}(0))$ or $(a_1(q) \leq a_2(q) \mid q = \text{optimize}(1))$ then *the EQUILINE does not pass through the convex hull*, otherwise *the EQUILINE passes through the convex hull.*

In the earlier case when the *EQUILINE* does not pass through the convex hull the required plan is at x=0 or x=1. This is shown in Figure 2.1.
In the later case we have to descend the hull and move to the base of it. This is done recursively using the below algorithm.

procedure Initialize()
Output: An interval [l, u] and plans p, q such that p = Min_B(optimize(l)) and q = Min_B(optimize(u)).
p := Min_B(optimize(random(0,1))
if$\langle a_1(p) = a_2(p) \rangle$ return [x,x,p,p]
if$\langle a_1(p) < a_2(p) \rangle$ return [x,1,p,Min_B(optimize(1))]
if$\langle a_1(p) > a_2(p) \rangle$ return [0,x,Min_B(optimize(0)),p]

procedure AffineApprox(l,u,p,q)
Input: $0 \leq l \leq u \leq 1$ such that p is optimal at l and q is optimal at u.
Output: The best affine approx w.r.t. the B metric in the interval [l,u].
if(l=u) return Min_B(p,q).
Let [l,x] = ClipRegion(p.[l,u],q)
r = Min_B(optimize(x)); if r is either p or q return r.
if$\langle a_1(r) > a_2(r) \rangle$ return AffineApprox(l,x,p,r)
if$\langle a_1(r) < a_2(r) \rangle$ return AffineApprox(x,u,r,q)
if$\langle a_1(r) = a_2(r) \rangle$ return r.

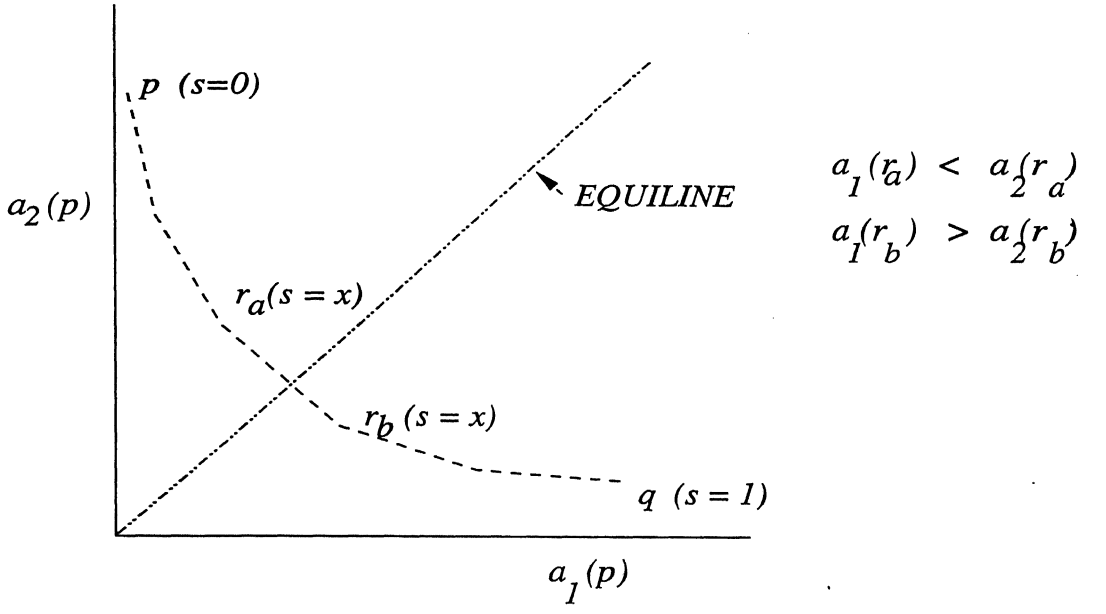


Figure 2.1: Algorithm for 2 parameter metric.

2.3 Properties of parameter space

A Database SQL query given in a non-procedural set oriented representation can be executed using different plans. Each plan is expressed by a query graph. The cost equation corresponding to each feasible plan for a given query can be represented by a point in the cost coordinate space. Hence a parameterized plan p of n parameters, corresponds to a point **A**(p) in the n+1 dimensional cost coordinate space.

Cost coordinate space CCS = $(a_0(\mathrm{p}),a_1(\mathrm{p}),\ldots,a_n(\mathrm{p}))$ $\forall$ p is a feasible plan.

Convex Hull A Convex Hull of a set of points is a boundary formed by the points such that all points in the set lie either inside or on the boundary and a segment joining any pair of points lies completely inside the polygon.

2.3.1 Convex hull

It has been shown by *Sumit Ganguly* in [Ganguly98] that *the convex hull of the set of points in the cost coordinate space corresponding to the set of feasible plans, defines the parametric optimal set of those plans.* That means all the plans lying on the convex hull represent the parametric optimal set *POS* of plans.

The complete convex hull represents the Parametric optimal set of plans when the parameters are assumed to be normalized ($0 \leq |x| \leq 1$). When the parameter value bounds is $0 \leq x \leq 1$, the parametric optimal set corresponds to only one half of the complete convex hull.

Two plans p and q lying on the convex hull which are adjacent to each other, are connected by the line having direction cosines **A**(p) - **A**(q). This line corresponds to the isocost plane of p and q in the parameter space each lying on either side of it. Thus two plans adjacent on the convex hull are neighbors in the parameter space.

All the plans which are adjacent to a plan p on the hull are neighbors of p in the parameter space, and try to bound the region of optimality R(p) on all directions from R(p). as shown in Figure 2.2.

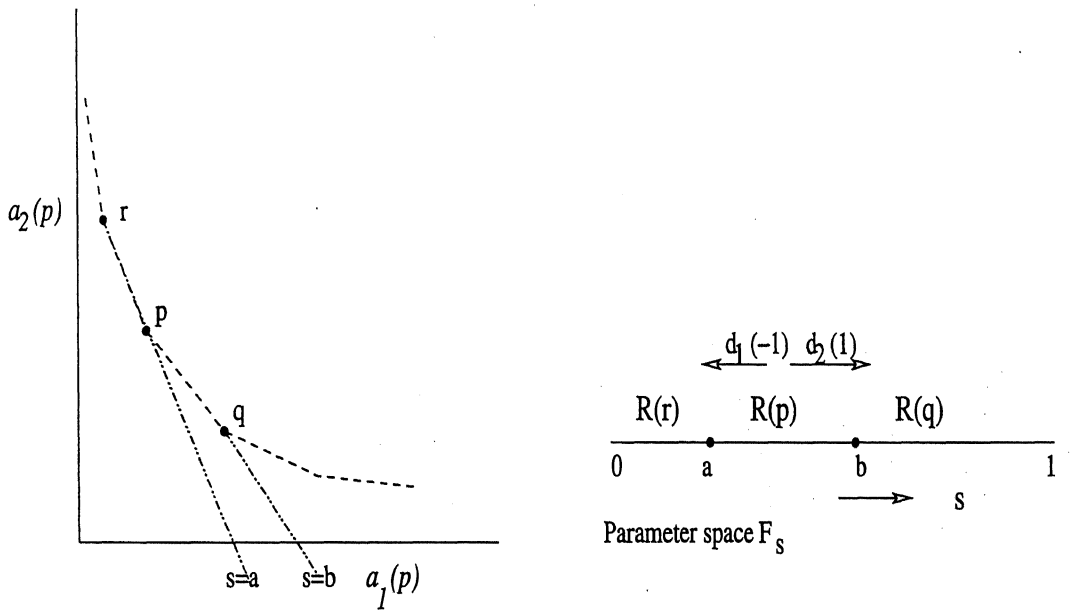


Figure 2.2: Neighborhood in parameter space and Adjacency on hull.

The region of optimality R(p) of a plan p is represented in the cost coordinate space by the range of the direction cosines of the normal to any line l that can be drawn passing through **A**(p), without intersecting any of the adjacent plans represented by the **A**(q). If the parameter space is $0 \leq x \leq 1$, then the region of optimality R(p) includes positive parameter values, hence the *direction cosines* of normal of l cannot be negative. That means the normal vector of every face of the

convex hull has positive direction cosines.

The convex hull corresponding to the positive parameter space is that half of the complete convex hull that lies below $l \cdot x + m \cdot y + n \cdot z + \cdots + k \cdot w = 1$ plane towards the origin, where $l, m, \ldots, k \geq 0$

The *EQUILINE* intersects the complete convex hull at two distinct points due to convexity of the complete convex hull.

The *EQUILINE* intersects the convex hull corresponding to positive parameter space either at one point or does not intersect at all.

The *EQUILINE* cannot intersect the convex hull at two points, that is the *EQUILINE* cannot pierce through our hull.

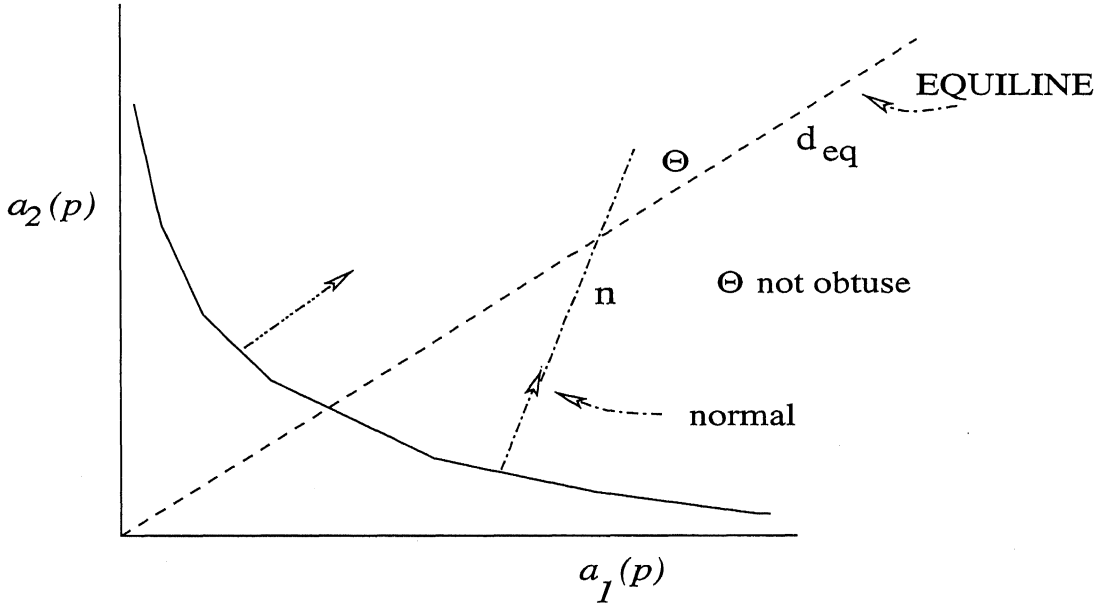


Figure 2.3: Intersection of EQUILINE with hull

Proof Let f_1 and f_2 be the two faces of the hull which the *EQUILINE* pierces. This is explained in Figure 2.3. That means the EQUILINE should make acute angle with inward normal vector of one face and obtuse angle with the inward normal vector of other face.

Let $\hat{n_1}$ and $\hat{n_2}$ be the normal vectors of the two faces. $\hat{d_{eq}}$ be the direction cosines of *EQUILINE*.

- Angle is obtuse implies $\hat{n} \cdot \hat{d_{eq}} < 0$.

- Angle is acute implies $\hat{n} \cdot \hat{d_{eq}} > 0$.

$$\hat{d_{eq}} = (1, 1, \ldots, 1).\hat{n} = (\hat{n_x}, \hat{n_y}, \ldots, \hat{n_w})$$

$$\hat{n} \cdot \hat{d_{eq}} = \hat{n_x} + \hat{n_y} + \ldots + \hat{n_w}$$

Since the direction cosines of the normal to a face of the hull are positive, $\hat{n} \cdot \hat{d_{eq}}$ can never be less than zero. Hence the angle between normal of face and *EQUILINE* cannot be obtuse. That is the *EQUILINE* cannot intersect the convex hull at two points.

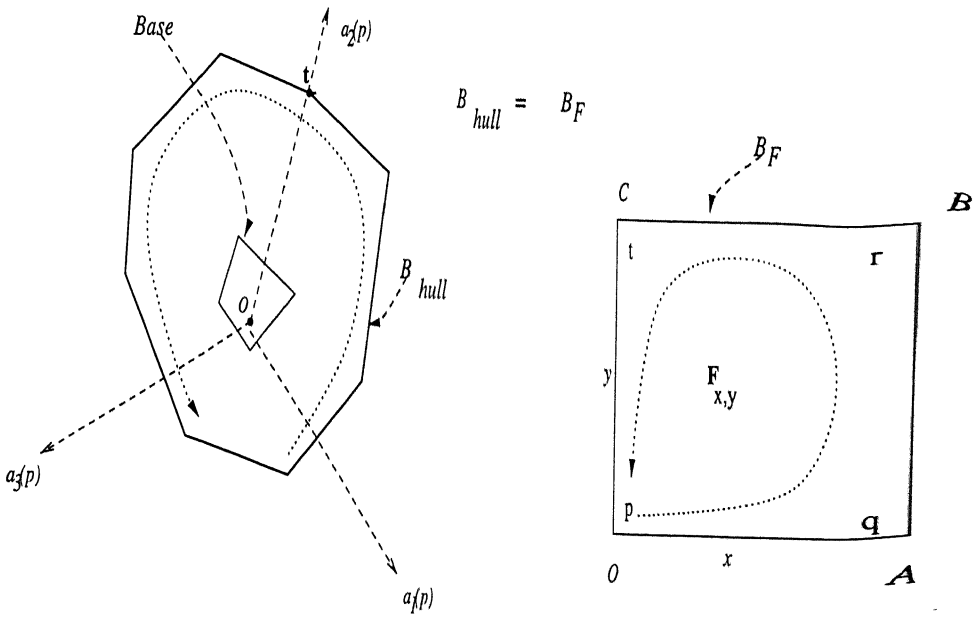


Figure 2.4: Boundary of hull and parameter space.

2.3.2 Boundary Theorem

The plans lying on the boundary B_{CH} *of the hull represents the parametric optimal set of plans on the boundary* B_F *of the parameter space* F.

Proof The plans lying on the lower part of the convex hull are adjacently bounded by some neighboring plans on all directions. That is in parameter space we cannot boundlessly increase the parameter value along some direction and still

remain inside R(p). While the plans lying on the boundary $\mathbf{B_{CH}}$ of the convex hull are not bounded by neighbors on all directions. So we can increase the parameter value boundlessly along any such direction **d** from **s**, and (s+**d**) belongs to R(p). Thus the plans lying on the boundary $\mathbf{B_{CH}}$ of the hull represents the parametric optimal set of plans on the boundary $\mathbf{B_F}$ of the parameter space **F**. This is depicted in the Figure 2.4.

2.3.3 Base corollary

If the EQUILINE does not intersect the convex hull, the base of the hull must also be a part of the boundary of the hull, and hence must lie on the boundary of the parameter space.

Let p be any plan on the convex hull in (a_0(p),a_1(p)) near the *EQUILINE* such that a_0(p) > a_1(p) and |a_0(p) - a_1(p)| = δ(a very small quantity). Any plan Q that is neighbor to P cannot lie in the rectangle region OAPB or infinite square region PB'IA' in the cost coordinate space, where O = (0,0), A = (a_0(p),0), P = (a_0(p),a_1(p)) and B = (0,a_1(p)) and B' = (∞,a_1(p)),A' = (a_0(p),∞) and I = (∞,∞). This is shown in the Figure 2.5.

Let E_1 and E_2 be two adjacent edges of the convex hull on which plan p lies. Edge cannot intersect the rectangle OAPB because of convexity of the hull. If the *EQUILINE* does not cut the convex hull, then one of the edge must lie in the region PB'IA'. But this is not possible because normal to any edge of convex hull cannot make an obtuse angle with the *EQUILINE*. Hence such a edge cannot exist. Thus such a plan p cannot have a adjacent neighbor on all directions if the hull is not cut by *EQUILINE*. The same applies when a_0(p) $\leq$ a_1(p). Thus plan p lies on the boundary of the convex hull, hence optimal along the boundary of the parameter space.

The above thesis can be applied for a higher dimensional case too. Let p be any plan on the convex hull in the (a_0(p),a_1(p),a_2(p)) cost coordinate space near the *EQUILINE* such that a_0(p) = max(a_i(p)). Any plan q that is neighbor to p must not lie in the 3 dimensional cube having the diagonal OP, O is origin and P = **A**(p). The normal to any face f_i on which plan p lies must not make an obtuse angle with

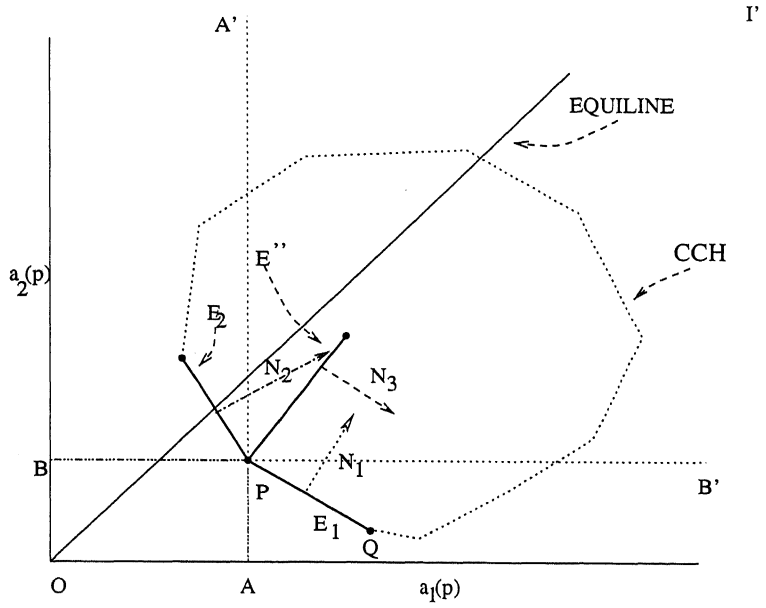


Figure 2.5: Base of Hull lies on the boundary of the hull.

the *EQUILINE*. That is any neighbor plan cannot lie in the 3 dimensional cube in cost coordinate space having AI as diagonal where A = **A**(p), I = (∞,∞,∞).

Thus q should either approach *EQUILINE* and cross it or go away from it. If the *EQUILINE* does not intersect the convex hull then q the neighbor of p must go away from *EQUILINE*. Thus p has no neighbor along such a plane. Thus p lies on the boundary of the convex hull.

Hence *if the EQUILINE does not intersect the convex hull the base of the hull must lie on the boundary of the hull, hence on the boundary of the parameter space.*

2.3.4 Neighbor Corollary

Consider the convex hull of parametric optimal plans in the cost coordinate space (a_0(p),a_1(p),a_2(p)) The parameter space is $\mathbf{F}_{x,y}$ 0 $\leq$ x,y $\leq$ 1. Any plan p which is neighbor to three plans p_1,p_2,p_3 must lie embedded between R(p_1),R(p_2) R(p_3) in the parameter space. The plan p which is optimal at the isocost point s_{iso} must be neighbor of the three plans. The plan p which is neighbor to the three plans must lie behind the plane formed by p_1,p_2 and p_3 on the convex hull in the cost coordinate

space.

Thus any plan p optimal inside the polyhedron formed by the points $\mathbf{s_1}$,$\mathbf{s_2}$,. . .,$\mathbf{s_n}$ *must lie behind the plane formed by plans optimal at these points in the n dimensional cost coordinate space.*

2.4 Approach for 3 parameter metric

In this section we will discuss the approach for applying PQO when the query cost metric is of the form:

$$B(p) = max(a_0(p), a_1(p), a_2(p))$$

$$PQODD : C(p, \mathbf{x}) = a_0(p) \cdot x_0 + a_1(p) \cdot x_1 + a_2(p) \cdot x_2,\ 0 \leq x_0, x_1, x_2 < \infty$$

The above PQODD can be expressed as 3 affine PQO problems given below:-

$$PQODD^0 : C^{(0)}(p, \mathbf{s}) = a_0(p) + a_1(p) \cdot s_1 + a_2(p) \cdot s_2\ 0 \leq s_1, s_2 \leq 1$$

$$PQODD^1 : C^{(1)}(p, \mathbf{s}) = a_1(p) + a_0(p) \cdot s_1 + a_2(p) \cdot s_2\ 0 \leq s_1, s_2 \leq 1$$

$$PQODD^2 : C^{(2)}(p, \mathbf{s}) = a_2(p) + a_0(p) \cdot s_1 + a_1(p) \cdot s_2\ 0 \leq s_1, s_2 \leq 1$$

The best affine approximation to PQODD is the better of the best affine approximations to $PQODD^0$, $PQODD^1$ and $PQODD^2$ respectively. We shall explain the general approach for finding the best affine approximation of $PQODD^0$.

In this case the convex hull is a surface in 3 dimensional cost coordinate space. The required plan lies on the base of this convex hull. The basic idea is to find 3 plans on the convex hull such that the plane formed by these three plans is intersected by the *EQUILINE*. The neighbor corollary says that any plan neighbor to three plans lies behind the plane formed by these three plans in the cost coordinate space. Also this neighbor plan has lower B(p) than atleast one of the three plans. Since normal to any face of even the partial hull cannot form an obtuse angle with the *EQUILINE*.

The region of optimality of this plan lies embedded within the regions of optimality of these three plans in the parameter space $\mathbf{F_{x,y}}$, so isocost point of these three plans in parameter space is a good choice to constrict the search space in parameter

space. Hence once we find such a triple of plans we can descend to the base of the convex hull using the isocost point of these three plans to constrict the search region in parameter space.

The algorithm first checks if such a triple of plans can be found among the plans optimal at the corners of the parameter space. If such a triple cannot be found out from the corners, we search the edges of parameter space for a plan using which an ideal triple can be constructed. This search of edges is further made efficient by searching only the prospective edge.

The three plans and the neighbor of these three plans p form a tetrahedron(a convex polyhedron) in the 3 dimensional cost coordinate space. The *EQUILINE* will either intersect the tetrahedron in 2 points(i.e., 2 faces) or will not intersect at all. Thus the descend hull would always yield a plane intersected by the *EQUILINE.* The algorithm recursively descends until no new plan is found out. There are *boundary cases* which can be appropriately handled.

The selection of prospective edge is done when the four plans optimal at the four corners cannot form a plane through which the *EQUILINE* passes. The prospective edge gives us the parameter space over which the partial convex hull be developed so that the *EQUILINE* passes through it. This is found using the intersection of *EQUILINE* with the plane of the three corner plans.

The basic idea is that *if the EQUILINE cuts the convex hull then we have to descend to the base of the hull, otherwise the base is also part of the boundary of the hull, the minimum B(p) plan lies on the base of the hull.*

2.5 Detailed description of Algorithm

This section gives the detailed description of the overall strategy and explains each of the algorithms used.

2.5.1 Algorithm Main

1. `Optimize at the four corners {O,A,B,C}`
   /* pmset = $\{p_1,p_2,p_3,p_4\}$, $\mathbf{P_{distinct}} \subset$ pmset.*/

2. if($|\mathbf{p_{distinct}}|$ == 1)
   {return the unique plan.}
   else
   {if($|p_{distinct}| < 3$)
   { edge = getprospective(p,q,r);
   $p_{distinct}$ = $p_{distinct}$ $\cup$ getsomeplan(edge);
   /* *make no of distinct plans to be 3,search on prospective edge.*/
   }
   }
   /* *To start with atleast 3 distinct plans are need to frame a plane* */

3. if(inside($p_{distinct}$) $\neq \Phi$)
   { $\mathbf{p_{distinct}}$ = do_boundary_checks();
   /* *Let f_1,f_2 be the two faces through which EQUILINE passes. Select the f_i which has the plan having minimum B(p)* */
   $p_{distinct}$ = face having B^{-1}(minimum(B(p_1),B(p_2),B(p_3),B(p_4)))
   goto step 4
   }
   else
   { /* *Intersecting plane cannot be framed from corner plans* */
   temp = getintplan($p_{distinct}$);
   /* *getintplan will search for a plan (to frame such a plane) on the prospective edge.*/
   if(inside(temp,p,q,r[,t]) == False) return search_on_boundary(temp);
   /* *inside(temp,p,q,r[,t])= False => The EQUILINE does not intersect the convex hull at all* */
   }

4. return Descend_Hull

Algorithm do_boundary_checks

This function does the checking for boundary cases. It checks for the following case:

- Check if $\{INT_1, INT_2\} \cap \{p_1, p_2, p_3, p_4\} \neq \Phi$. Return INT if one of the vertex lies on the *EQUILINE*.

- Rest all other condition checks are taken care of in Descend_Hull algorithm

For the case 1 it returns the INT_1 or INT_2.

Description Main

In step 1 we optimize at the four corners of the parameter space. Let pmset be the multiset of plans p_1, p_2, p_3 and p_4, optimal at the corners and let $\mathbf{p_{distinct}}$ be the set of distinct plans among these. Since a plane in three dimensional cost coordinate space is uniquely defined by three distinct plans, we check the cardinality of the $\mathbf{p_{distinct}}$ set.

If the cardinality of the set $\mathbf{p_{distinct}}$ is one or in other words only a single plan is optimal at the four corners, then by the property of convexity of optimality region only one plan is optimal on the whole parameter space. For uniquely defining a plane requires three plans, so if the cardinality of set $\mathbf{p_{distinct}}$ is less than three we search for some more plans on the prospective edge. A prospective edge is the boundary edge in the parameter space, on which the prospects of finding a plan of our interest are bright.

The *EQUILINE* intersects the tetrahedron either at two points or does not intersect at all. The inside function returns true if the *EQUILINE* intersects in Δpqr for any three plans p,q,r. We also perform boundary conditions check to deal with the case the *EQUILINE* touches the tetrahedron. If the EQUILINE ideally intersects the tetrahedron in two faces, the face(s) which have the plan having minimum B(p), that is B^{-1}(minimum(B(p_1),B(p_2),B(p_3),B(p_4))) is selected.

If the tetrahedron is not intersected by the *EQUILINE* we search for a possibility by developing the partial Hull along the prospective direction or in other words search on prospective edge of parameter space. Procedure getintplan does the above task. If *getintplan* returns a plan which does not form an intersecting plane it implies that whole of convex hull is not intersected by the *EQUILINE*. Since the boundary of the convex hull represents the parametric optimal plans along the boundary of

the parameter space we search the boundary.

If we could construct a triple of three distinct plans through which the *EQUILINE* passes we descend to the base of the convex hull using the *Descend_Hull* algorithm.

2.5.2 Search_on_boundary

Input: p_{base}

The *getintplan* when returns unsuccessfully, returns a plan in the base of the convex hull. Since in this case the convex hull lies on one side of the *EQUILINE* and so the boundary of the convex hull also includes the base. That is the base of convex hull is also present on the boundary of the parameter space. So search for the best plan in term of B(p) metric in the neighborhood of p_{base} on the appropriate boundary edge.

2.5.3 Algorithm getprospective

```
Input p,q,r
Output set of tuples {p1,p2}
result = Φ;
ss = nearest({p,q,r});
/* {p,q} ∈ ss */
for(each {p,q} ∈ ss)
{ if({p,q} forms a real edge of OABC)
result = result ∪ {p,q};
}
return result;
```

■ *Description getprospective*

This function returns the prospective edge defined by the plans optimal at the corners. The definition of *prospective edge* is in Figure 2.6. It finds the nearest edge to the *EQUILINE* and if it corresponds to some *real edge* on the parameter space it

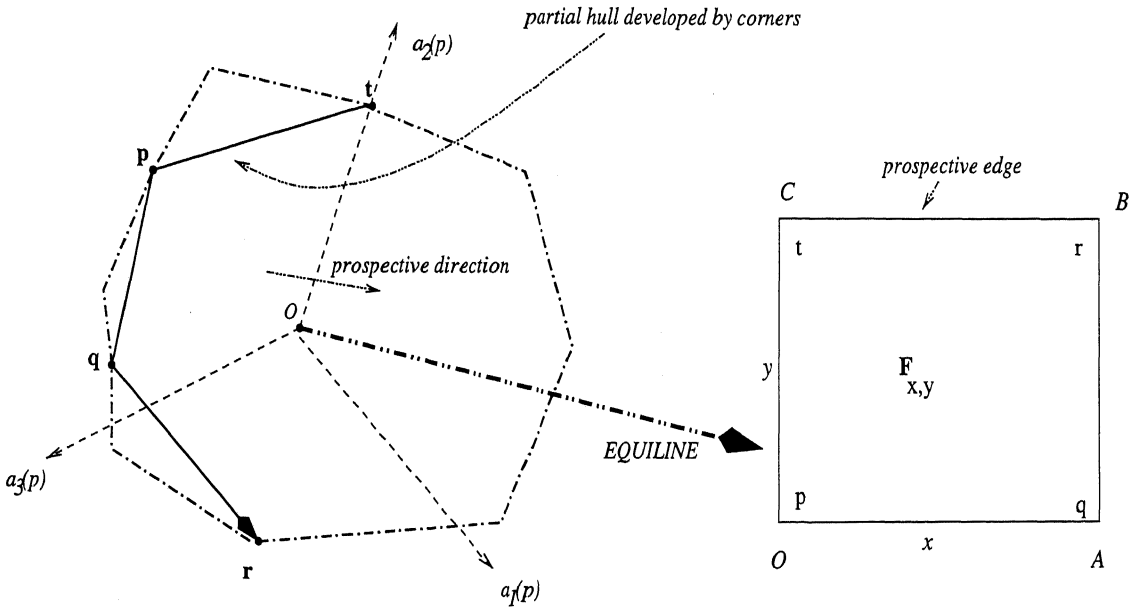


Figure 2.6: Defining Prospective Edge.

returns that edge. If the nearest edge is not real edge then it returns null. Any edge is real if it corresponds to some existing boundary edge of the parameter space.

2.5.4 Algorithm getintplan

```
Input edgeset, All_corners
Output p_int
prosedgeset = getprospective(edgeset)
for(each pros ∈ prosedgeset)
{
temp.s = Isocost_edgei(pros.p,pros.q);
/* pros.p and pros.q are plans optimal at the corners of the edge i */
temp.plan = optimize(temp.s);
if(temp.plan ∈ {pros.p,pros.q})
{ remove(prosedgeset,pros)
prosedgeset = prosedgeset ∪ edge(temp.plan,nplan)
/* where nplan ∈ pros and nplan ≠ temp.plan) */
```

```
continue }
if(inside(ss,temp.plan) | ss ⊂ All_corners and |ss| = 2 )
{ return temp.plan }
tempedgeset = the two sub edges partitioned at temp.s
remove(prosedgeset,pros)
prosedgeset = prosedgeset ∪ getprospective(tempedgeset)
}
return temp.plan
```

Description getintplan

The getintplan searches for an interesting plan on a *prospective edge* of the parameter space. Once it finds out some prospective edge it optimizes at the isocost point of the two plans optimal at the end points of the edge. This is a non recursive procedure and terminates only if no new plan is found and the prospective parameter space is completely traversed. It adopts a *divide and conquer* strategy. Suppose p,q are the optimal plans at the end points of the prospective edge, and say p_{iso} is the optimal plan at the isocost point of the two plans p,q. We make a decision of conquering which half of the prospective edge. If the convex hull lies to one side of the *EQUILINE* the function *getintplan* returns the plan which is at the base as it recurses down.

Proof for getintplan algorithm

By traversing the prospective edge we can get a plan to frame a plane intersected by the EQUILINE, which is of interest to us.

Proof When the partial hull that has been developed is not intersected by the *EQUILINE*, then we need to further develop the *partial hull* between the two plans that are spatially nearer to the *EQUILINE*. That is the prospective direction and developing the hull along such direction would be favorable to frame a plane intersected by the *EQUILINE*. The prospective edge is that edge of the parameter space boundary, corresponding to such a set of plans which are optimal at the ends of the

edge.

If the *EQUILINE* does not intersect the convex hull at all, that means the hull lies to one side of the *EQUILINE*, then we cannot frame such a plane at all. The divide and conquer approach will in that case would descend to the base of the hull which also lies on the boundary of the hull, that is on the prospective edge of the boundary of the parameter space.

2.5.5 Algorithm Descend_Hull

```
Input p,q,r,t,s1,s2,s3,s4;
Output plan;
if(t ≠ Φ)
{
/* To select the three plans intersected by EQUILINE out of two intersecting faces
formed by corners */
temp.s = s4;
temp.plan = t;
}
else
{
temp.s = ISOCOST(p,q,r);
temp.plan = optimize(temp.s);
}
while(not(temp.plan ∈ {p,q,r})
{
/*Loops till some new plan is discovered*/
{p,q,r} = process_descend(p,q,r,temp.plan,s1,s2,s3,temp.s);
if(|{p,q,r}| == 0) return main(rectangle(s1,s2,s3));
/*If the EQUILINE lies in the plane of a face or touches the tetrahedron call main
at that point new boundary of the hull*/
temp.s = ISOCOST(p,q,r);
temp.plan = optimize(temp.s);
```

}

/*Return the plan having minimum value of B(p) at the base*/

return best(p,q,r);

Algorithm process_descend

Input p,q,r,$p_{isocost}$,s_1,s_2,s_3,$s_{isocost}$

Output {p,q,r}

{INT_1,INT_2} = intersection(p,q,r,$p_{isocost}$,EQUILINE).

if(INT_2 $\in$ {p,q,r,$p_{isocost}$}) return INT_2;

/* INT_i is any vertex return that INT_i */

if(INT_2 = Φ)

{ /* The EQUILINE touches tetrahedron, generate set of new appropriate plans */

return Φ;

}

if(INT_1 $\in$ Δpqr and INT_2 $\in$ Δpqr) {

return Φ;

/* The EQUILINE lies in the plane of a face of tetrahedron */

}

if(INT_1 $\in$ face INT_2 $\in$ face2)

{ return face2; }

else

{

A_1 : {p,q,optimize(isocost(p,$p_{isocost}$,r))}

A_2 : {p,r,optimize(isocost(p,$p_{isocost}$,q))}

if(inside(A_1)) return A_1;

if(inside(A_2)) return A_2;

else return {p,q,isocost(p,q,r)}

/*This case is handled subsequently, a EQUILINE touching the face is returned*/

}

Description Descend_Hull

The Descend_Hull algorithm takes 3 distinct plans as input, such that the triangular plane through them in cost coordinate space is intersected by *EQUILINE*. This is depicted in Figure 2.7. Three points $\{s_1,s_2,s_3\}$ in the parameter space $\mathbf{F_{x,y}}$ are also taken such that each
$s_i \in$ R(pl) | pl $\in$ {p,q,r}. Any plan $p_{isocost}$ that is optimal at the isocost of these three plans must lie on the convex hull behind the plane formed by them in cost coordinate.

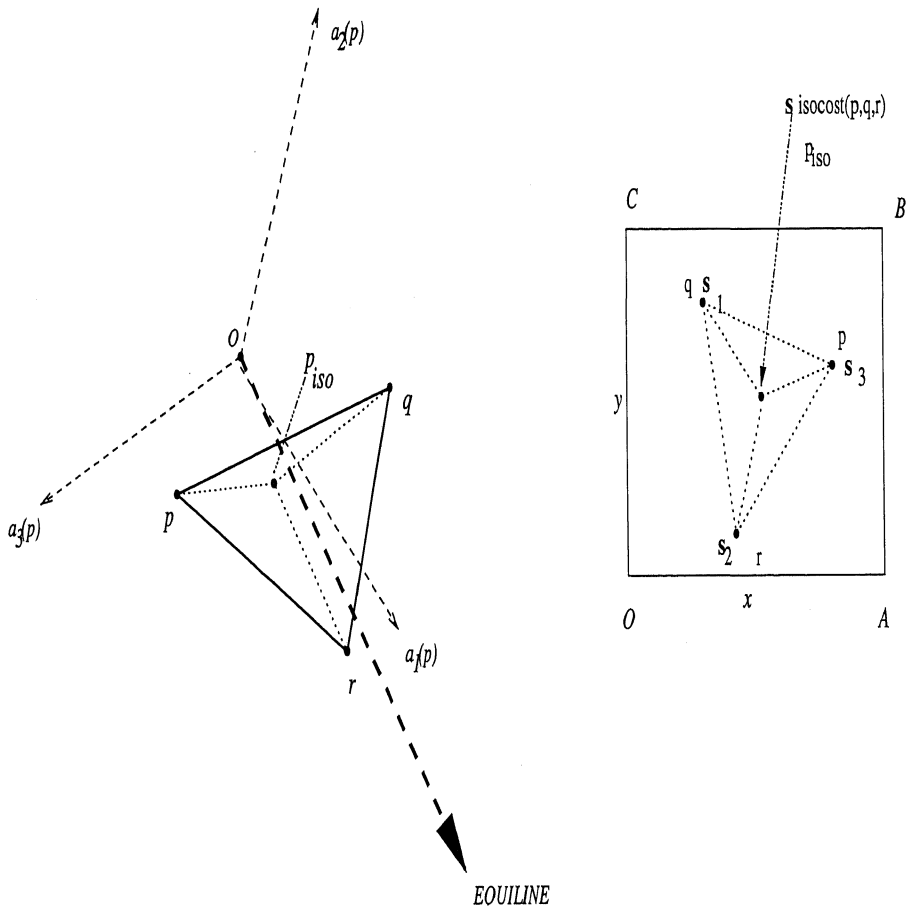


Figure 2.7: Descend Hull Algorithm.

If $p_{isocost}$ lies on the EQUILINE then $p_{isocost}$ is the required plan. Otherwise it is guaranteed that $B(p_{isocost}) < max(B(p),B(q),B(r))$, when EQUILINE intersects the

Δpqr

{p,q,r,$p_{isocost}$} forms a tetrahedron(a convex polyhedron). *EQUILINE* intersects the tetrahedron at 2 points {INT_1,INT_2}. {INT_1,INT_2} $\cap$ {p,q,r} = Φ is guaranteed. Let INT_1 be lying on the plane of {p,q,r}, INT_1 can lie either on edges or on triangular face Δpqr.

1. INT_1 lies inside on the face of Δpqr

   - $INT_2 \neq \Phi$

     $p_{isocost}$ lies on *EQUILINE* implies $INT_2 = p_{isocost}$, then $p_{isocost}$ is required one. Otherwise INT_2 can lie on any edge or adjacent face.

     If INT_2 lies on a face then we replace the plan B^{-1}(max(B(p),B(q),B(r)) by $p_{isocost}$ which in the resultant face is guaranteed to be intersected by *EQUILINE*.

     If INT_2 lies on the say (p,$p_{isocost}$) edge find the optimize(isocost(p,$p_{isocost}$,q)) (or optimize(isocost(p,$p_{isocost}$,r))) and do above checks considering the tetrahedron {p,optimize(isocost(p,$p_{isocost}$,r)),q,r} or {p,optimize(isocost(p,p_{isocos

2. INT_1 lies on the edge of Δpqr.

   - $INT_2 \neq \Phi$

     $p_{isocost}$ lies on *EQUILINE* implies $INT_2 = p_{isocost}$, then $p_{isocost}$ is required one. Otherwise INT_2 can lie on any edge or adjacent face.

     If INT_2 lies on a face then we replace the plan B^{-1}(max(B(p),B(q),B(r)) by $p_{isocost}$ which in resultant face is guaranteed to be intersected by *EQUILINE*.

     If INT_2 lies on the say (p,$p_{isocost}$) edge find the optimize(isocost(p,$p_{isocost}$,q)) (or optimize(isocost(p,$p_{isocost}$,r))) and do above checks considering the tetrahedron {p,optimize(isocost(p,$p_{isocost}$,r)),q,r} or {p,optimize(isocost(p,p_{isocos

   - $INT_2 = \Phi$

     Consider the rectangular parameter space $O_1A_1B_1C_1$ formed by {s_1,s_2,s_3} in the parameter space.
     return main(O_1,A_1,B_1,C_1)

3. Both INT_1 and INT_2 lie on the same triangular face of Δpqr.

   Consider the rectangular parameter space $O_1A_1B_1C_1$ formed by $\{s_1,s_2,s_3\}$ in the parameter space.
   return main(O_1,A_1,B_1,C_1)

Proof for Descend_Hull algorithm

The Descend_Hull algorithm tries to frame a plane in a n dimensional cost coordinate space through which the EQUILINE passes and tries to descend to the base of the hull.

Proof Suppose p,q and r be three plans in 3 dimensional cost coordinate space (a_0(p),a_1(p),a_2(p)). Any plan p_{neigh} that is neighbor to the three plans must lie behind the plane formed by p,q,r, towards the origin. Plans p,q,r,p_{neigh} form a tetrahedron(a convex polyhedron), which is the **partial convex hull**. Since none pf the face can make an obtuse angle with the *EQUILINE*, p_{neigh} should lie behind the plane. In case p_{neigh} lies above the plane formed by p,q,r then the other three faces of the tetrahedron will be making an obtuse angle.

Any plan that is neighbor to the three plans must lie embedded within the region of optimality of the three plans in the parameter space. Any straight line that intersects a convex polyhedron must intersect exactly at two points(faces) or does not intersect at all, taking care that the line does not touch the polyhedron (a boundary case with one point of intersection).

Thus descend hull algorithm safely descends to the base of the hull.

2.5.6 Algorithm nearest

Input: p,q,r;
Output: result
PL = plane(p,q,r);
INT =intersection(*EQUILINE*,PL);
for(each ss $\subset$ {p,q,r}, |ss| = 2)
{ dv = direction vector of line formed by ss

if((dprod =(dv·INT)·(dv·w)) $\geq$ 0, w $\in$ {p,q,r} and w $\notin$ ss)
result = result $\cup$ ss
}
return result;

- *Algorithm inside*

Input: p,q,r;
Output: True or False
result = nearest(p,q,r);
if(|result| == 3) return True; else False;

- *Description nearest and inside*

The function nearest finds the edge which is nearest to the *EQUILINE* of the three edges of triangle formed by three plans. The function finds the intersection point of the *EQUILINE* with the plane formed by the three plans, say INT. Let **dv** be the direction cosines of edge **A**(q)**A**(r). If product of dot products of **dv** with coordinates of **A**(p) and dot product of **dv** with INT is negative than both p and *INT* lie on opposite sides and this edge **A**(q)**A**(r) is the nearest edge.

The function finds whether INT is inside the Δpqr. It calls *nearest* for each two vertex subset of {p,q,r}, and if each vertex and INT lies on the same side with respect to other two vertices(other edge) then the cardinality or number of tuples returned by nearest would be *zero*. Hence INT lie inside Δpqr.

2.5.7 Analysis of Boundary conditions

Consider the case when we are having three distinct plans forming a tetrahedron in the cost coordinate space. Let us assume that the *EQUILINE* intersects the tetrahedron. The following cases arise:

1. The EQUILINE *does not intersect or touch* the tetrahedron.

   Find the prospective edge and getintplan on it.

2. The EQUILINE *touches* the tetrahedron.

   Here only one point of intersection INT_1 is there. $INT_2 = \Phi$. INT_1 can be a vertex or and edge. If INT_1 is vertex then return INT_1, otherwise

3. The EQUILINE *intersects* the tetrahedron at 2 points.

   Here the following are the possibilities:

   (a) INT_1 is vertex

      i. INT_2 is vertex return INT_1

      ii. INT_2 is face return INT_1

      iii. INT_2 is edge return INT_1

   (b) INT_1 is face

      i. INT_2 is vertex return INT_2

      ii. INT_2 is face return the face containing the plan B^{-1}(min(B(p),B(q),B(r),B(t)))

      iii. INT_2 is edge

      search for a new plan to replace in place of t (or optimize(iso(p,q,r))).
      The new plan can be t_{new} = optimize(iso(p,q,t)) or t_{nnew} = optimize(iso(p,r,t)) and check is intersecting plane be constructed using plane(p,q,t_{nnew}) or plane(p,r,t_{new})).

   (c) INT_1 is edge

      i. INT_2 is vertex return INT_2

      ii. INT_2 is face

      search for a new plan to replace in place of t (or optimize(iso(p,q,r))).
      The new plan can be t_{new} = optimize(iso(p,q,t)) or t_{nnew} = optimize(iso(p,r,t)) and check is intersecting plane be constructed using plane(p,q,t_{nnew}) or plane(p,r,t_{new})).

      iii. INT_2 is coplanar edge Both INT_1 and INT_2 lie in the same plane ie; Δ pqr
      Consider the rectangular parameter space $O_1A_1B_1C_1$ formed by $\{s_1,s_2,s_3\}$ in the parameter space.
      return main(O_1,A_1,B_1,C_1)

iv. INT_2 is non coplanar edge

search for a new plan to replace in place of t (or optimize(iso(p,q,r))). The new plan can be t_{new} = optimize(iso(p,q,t)) or t_{nnew} = optimize(iso(p,r,t)) and check is intersecting plane be constructed using plane(p,q,t_{nnew}) or plane(p,r,t_{new})).

2.5.8 Miscellenious Algorithms

The following are the miscellaneous functions which have been also used.

1. *optimize* finds the plan(s) optimal at any point s in the parameter space.
2. *Isocost* finds the isocost point $\mathbf{s_{isocost}}$ of a set of plans.
3. *rectangle* returns the rectangular parameter space formed by 3 points s_1,s_2,s_3 in the parameter space.
4. *intersection* finds the intersection of EQUILINE with the plane formed by 3 plans in cost coordinate space plane($\mathbf{A}$(p),$\mathbf{A}$(q),$\mathbf{A}$(r)).
5. *getsomeplan* finds out some plan on the given edge of parameter space.

Chapter 3

Algorithm for n parameters

This chapter discusses the techniques for extending the algorithm described in previous chapter to n dimensional complex query metric. The baseline of the whole approach remains the same. First we discuss the extensions to the basic approach to solve for 4 parameter query metric. Then we give a generalized algorithm for n parameter query metric.

3.1 Extending to 4 parameter metric

Consider the following query metric in 4 parameters.

$$B(p) = max(a_0(p), a_1(p), a_2(p), a_3(p))$$

$$PQODD : C(p, \mathbf{x}) = a_0(p) \cdot x_0 + a_1(p) \cdot x_1 + a_2(p) \cdot x_2 + a_3(p) \cdot x_3,\ 0 \leq x_0, x_1, x_2, x_3 < \infty$$

The four affine parameterized PQO problems are of the following type:

$$PQODD^i : C^{(i)}(p, x, y, z) = a_0(p) + a_1(p) \cdot x + a_2(p) \cdot y + a_3(p) \cdot z,\ 0 \leq x, y, z \leq 1$$

Parameter space is $\mathbf{F}_{(\mathbf{x},\mathbf{y},\mathbf{z})}$, where $0 \leq$ x,y,z ≤ 1
The cost coordinate space is $\mathbf{A}$(p) =(a_0(p),a_1(p),a_2(p),a_3(p)). The parameter space is a cube in three dimensions. The cube is bounded by six two-dimensional faces $\{f_1, f_2, \ldots, f_6\}$. We have eight corner points CORNERS = $\{O, C_1, C_2, \ldots, C_7\}$.

$$EQUILINE : a_0(p) = a_1(p) = a_2(p) = a_3(p)$$

The cost coordinate space is four dimensional space . The convex hull in the four dimensional cost coordinate space represents the parametric optimal set of plans **POS**.

We need four distinct plans to define a three dimensional plane in the cost coordinate space. We try to frame a three dimensional plane in cost coordinate space, through which the *EQUILINE* passes. If such a plane can be framed by a subset of four plans optimal at the corner points we proceed to Descend_Hull algorithm, otherwise we extend the partial convex hull in the *prospective direction*, by searching for an interesting plan on the prospective face f_{pros}.

The four plans and the neighbor of these four plans p, form a polyhedron (convex) in four dimensions, bounded by three dimensional faces. The *EQUILINE* will either intersect the polyhedron at two points (convexity of polyhedron) or does not intersect it at all. Hence the descend hull would always yield a face (three dimensional plane) having lower B(p) value out of all the vertices of the polyhedron. Also more than two plans cannot be collinear in the cost coordinate space, since we are choosing the plans to be lying on the convex hull.

3.1.1 Computing the best affine approximation

This algorithm also consist of the following main sub algorithms.

- Algorithm *main*
- Algorithm *Descend_Hull*
- Algorithm *process_descend*
- Algorithm *getprospective*
- Algorithm *getintplan*
- Algorithm *nearest*

The overall approach is the same, only the notion and method of finding the *prospective face* has slight modifications.

3.1.2 Algorithm Main

1. Optimize at the corner points {0,C_1,C_2,...,C_7}
   /* *pmset* = { p_1,p_2,...,p_8} *be the multiset of the eight plans.*/
   /* Let **Pdistinct** ⊂ pmset which are all distinct.*/

2. if(| $p_{distinct}$ | == 1)
   {return *the unique plan.*}
   else
   {if(| $p_{distinct}$ | < 4)
   { face = getprospective(pmset);
   $p_{distinct}$ = $p_{distinct}$ ∪ getsomeplan(face);
   /* *make no of distinct plans to be 4,by searching on prospective face.*/
   }
   }

3. if(inside($p_{distinct}$) ≠ Φ)
   /* *a 3 dimensional intersecting plane can be framed using corners* */
   { $p_{distinct}$ = do_boundary_checks();
   /* *Let* $face_1$,$face_2$ *be the two 3 dimensional faces through which EQUILINE passes. Select the* $face_i$ *which has the minimum B(p) plan* */
   select B^{-1}(minimum(B(p_1),B(p_2),...,B(p_8)))
   goto step 4
   }
   else
   { /* *Intersecting plane cannot be framed from corner plans* */
   temp = getintplan();
   /* *getintplan will search for a plan (to frame such a plane) on the prospective face.*/
   if(inside(temp,$p_{distinct}$) == False) return search_on_boundary(temp);
   /* *inside(temp,*$p_{distinct}$*)= False => The EQUILINE does not intersect the convex hull at all* */
   }

4. return Descend_Hull.

Description Main

In step 1 we optimize at the eight corner points of the parameter space. Let pmset be the multiset of plans p_1,p_2,p_3,...,p_8, optimal at the corners and let $\mathbf{p_{distinct}}$ be the set of distinct plans among these.

We check the cardinality of the $\mathbf{p_{distinct}}$ set. If the cardinality of the set $\mathbf{p_{distinct}}$ is one, then we return the only plan as in 3 parameters case. And if the cardinality of set $\mathbf{p_{distinct}}$ is less than four we search for some more plans along the *prospective direction*, that is on the prospective face f_{pros} of the parameter space. A prospective face is the bounding face of the parameter space, on which the prospects of finding a plan of our interest are better.

The *EQUILINE* will intersect the 4 dimensional polyhedron either at two points or does not intersect at all. We also perform boundary conditions check to deal with the case the *EQUILINE* touches the polyhedron. If the *EQUILINE* ideally intersects the polyhedron in two faces, the face(s) which have the plan having minimum B(p), that is B^{-1}(minimum(B(p_1),B(p_2),B(p_3),B(p_4),B(p_5))) is selected.

If the polyhedron is not intersected by the *EQUILINE* we search for a possibility by developing the *partial Hull along the prospective direction* by searching on prospective face in parameter space. Procedure getintplan does the above task. If *getintplan* returns a plan which does not form an intersecting plane it implies that whole of convex hull is not intersected by the *EQUILINE*. Since the boundary of the convex hull represents the parametric optimal plans along the boundary of the parameter space we search the boundary.

If we could construct a quadruple of four distinct plans through which the *EQUILINE* passes we descend to the base of the convex hull using the *Descend_Hull* algorithm.

Algorithm do_boundary_checks

The polyhedron formed by plans in $p_{distinct}$ is convex and hence *EQUILINE* will either intersect at one point INT_1 (boundary conditions touch) or two points INT_1,INT_2 or does not intersect at all the polyhedron($\mathbf{p_{distinct}}$).

This function does the checking for boundary cases. It checks for the following case:

- Check if $\{INT_1,INT_2\} \cap \mathbf{p_{distinct}} \neq \Phi$.
- Rest all other condition checks are taken care of in Descend_Hull algorithm

For the case 1 it returns the INT_1 or INT_2.

3.1.3 Algorithm getprospective

```
Input p_distinct /*Set of distinct plans */
Output set of tuples {p,q,r,t}
result = tresult = Φ;
For(each face f_i of the parameter space)
{
plansi = plans(f_i);
/* plansi is multiset of plans optimal at the corners of the face f_i */
for (each ss ⊂ plansi | |ss|=3)
{
pother = a distinct plan belonging to a corner not on this face.
stemp = nearest(ss,pother);
/* {p,q,r} ∈ stemp */
for(each {p,q,r} ∈ stemp )
{ if({p,q,r} forms a adjacent corners of this face f_i)
{ tresult = tresult ∪ {p,q,r};}
}
}
for(each f_i)
```

```
{ if (each triple adjacent corners are nearest)
result = result ∪ f_i;
}
return result;
```

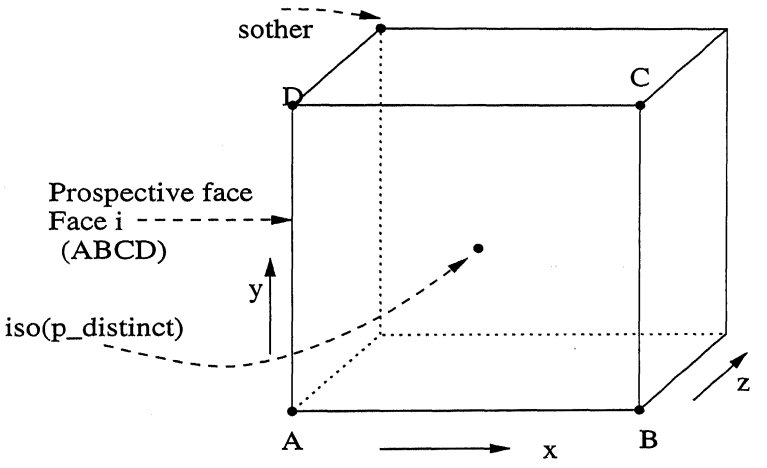


Figure 3.1: Prospective face in 4 parameter metric.

- *Description getprospective*

This function returns the prospective face of the parameter space on which the prospects of finding the plan of interest are brighter. For each face of the parameter space, we consider a subset of 3 adjacent plans on the face and a plan *pother* not belonging to this face. If the nearest face of the tetrahedron of these four plans corresponds to some real face, then add the three adjacent plans to the result set tresult. If all such three adjacent triple of plans of a face are real, then the particular face of the parameter space is the prospective face.

3.1.4 Algorithm getintplan

```
Input FaceSet, All_corners
Output p_int
prosfaceset = getprospective(Faceset)
for(each pros ∈ prosfaceset)
```

```
{
temp.s = Isocost_fi(pros.plans);
/* pros.plans = {pros.p1,pros.p2,pros.p3,pros.p4} */
temp.plan = optimize(temp.s);
if(inside(ss,temp.plan) |(for some ss ⊂ All_corners and |ss| == 3)
{return temp.plan }
tempprosset = the four subfaces partitioned at temp.s
remove(prosfaceset,pros)
prosfaceset = prosfaceset ∪ getprospective(tempprosset)
}
return temp.plan
```

- *Description getintplan*

The getintplan first finds the prospective face f_i. Find the isocost point of the four plans at the corners. Let $p_{isocost}$ be the plan optimal at the isocost point. If a intersecting plan can be formed using the plan $p_{isocost}$ return the plan $p_{isocost}$ other wise find subface out of the four subface which is prospective. Recurse till the parameter space on the face is complete. Repeat this *divide and conquer strategy* for each prospective face belonging to prosfaceset.

3.1.5 Algorithm Descend_Hull

```
Input p,q,r,t,u,s1,s2,s3,s4,s5;
Output plan;
if(u ≠ Φ)
{ temp.s = s5; temp.plan = u; }
else
{ temp.s = ISOCOST(p,q,r,t); temp.plan = optimize(temp.s);}
while(not(temp.plan ∈ {p,q,r,t})
{
{p,q,r,t} = process_descend(p,q,r,t,temp.plan,s1,s2,s3,s4,temp.s);
```

```
if(|{p,q,r,t}| == 0) return main(cube(s1,s2,s3,s4));
temp.s = ISOCOST(p,q,r,t);
temp.plan = optimize(temp.s);
}
return best(p,q,r,t);
```

3.1.6 Algorithm process_descend

Input p,q,r,t,$p_{isocost}$,s_1,s_2,s_3,s_4,$s_{isocost}$
Output {p,q,r,t}
{INT_1,INT_2} = intersection(p,q,r,t,$p_{isocost}$,EQUILINE).
if($INT_2 \in$ {p,q,r,t,$p_{isocost}$}) return INT_2;
/* *INT_i is any vertex return that INT_i* */
if(INT_2 = Φ)
{ /* *EQUILINE touches polyhedron, generate set of new appropriate plans* */
return Φ;
}
if($INT_1 \in$ tetra(pqrt) and $INT_2 \in$ tetra(pqrt)) {
return Φ;
/* *The EQUILINE lies in the tetrahedral plane of a face of polyhedron* */
}
if($INT_1 \in$ face $INT_2 \in$ face2)
{ return face2; }
else
return Φ.
}

3.1.7 Algorithm nearest

Input: p,q,r,t;
/* *Four plans forming a 3 dimensional plane* */
Output: result

```
PL = plane(p,q,r,t);
/* PL is 3 dimensional plane */
INT =intersection(EQUILINE,PL);
for(each ss ⊂ {p,q,r,t}, |ss| = 3 )
{ dv = direction vector of 2 dimensional plane formed by ss
if((dprod =(dv·INT)·(dv·w)) ≥ 0, w ∈ {p,q,r,t} and w ∉ ss)
result = result ∪ ss
}
return result;
```

3.1.8 Algorithm inside

```
Input:  p,q,r,t;
Output:  True or False
result = nearest(p,q,r,t);
if(|result| == 4) return True; else False;
```

- *Description nearest and inside*

The function nearest finds the face which is nearest to the *EQUILINE* of the four faces of tetrahedron formed by four plans. The function finds the intersection point of the *EQUILINE* with the 3 dimensional plane formed by the four plans, say INT. Let **dv** be the direction cosines of 2 dimensional face $\mathbf{A}(q)\mathbf{A}(r)\mathbf{A}(t)$. If product of dot products of **dv** with coordinates of $\mathbf{A}(p)$ and dot product of **dv** with INT is negative than both p and *INT* lie on opposite sides and this face $\mathbf{A}(q)\mathbf{A}(r)\mathbf{A}(t)$ is the nearest face.

The function finds whether INT is inside the tetrahedron pqrt. It calls *nearest* for each three vertex subset of {p,q,r,t}, and if each vertex and INT lies on the same side with respect to other three vertices(other face) then the cardinality or number of tuples returned by nearest would be 4. Hence INT lie inside tetrahedron pqrt.

3.1.9 Miscellaneous Algorithms

The following are the miscellaneous functions which have been also used. *Optimize* finds the plan(s) optimal at any point s in the parameter space. *Isocost* finds the isocost point $\mathbf{s_{isocost}}$ of a set of plans. *Cube* returns the cube of parameter space formed by 4 points s_1,s_2,s_3,s_4 in the parameter space. *Intersection* finds the intersection of EQUILINE with the plane formed by 4 plans in cost coordinate space plane(A(p),A(q),A(r),A(t)). *getsomeplan* finds out some plan on the given edge of parameter space.

3.2 Extending to n+1 parameter metric

The above approach which has been extended for 4 parameter case, can now easily be made to work for any n-parametric cost equation. Consider the following query metric in n+1 parameters.

$$B(p) = max(a_0(p), a_1(p), a_2(p), \ldots, a_n(p))$$

$$PQODD : C(p, \mathbf{x}) = a_0(p)\cdot x_0 + a_1(p)\cdot x_1 + a_2(p)\cdot x_2 + \ldots + a_n(p)\cdot x_n,\ 0 \leq x_0, x_1, x_2, \ldots, x_n < \infty$$

The n+1 affine parameterized PQO problems are of the following type:

$$PQODD^i : C^{(i)}(p, s_1, s_2, \ldots, s_n) = a_0(p) + a_1(p)\cdot s_1 + a_2(p)\cdot s_2 + \ldots + a_n(p)\cdot s_n,\ 0 \leq s_1, s_2, \ldots, s_n \leq 1$$

The cost coordinate space is A(p) =(a_0(p),a_1(p),a_2(p),...,a_n(p)).

$$EQUILINE : a_0(p) = a_1(p) = \ldots = a_n(p)$$

The parameter space consists of a hypercube in n dimensions. This hypercube is bounded by 2n (n-1)-dimensional faces {f_1,f_2,...,f_{2n}}. We have 2^n corner points CORNERS = {O,C_1,C_2,...,C_k}, k = 2^n - 1. The cost coordinate space is n+1 dimensional space . The convex hull in the n+1 dimensional cost coordinate space represents the parametric optimal set of plans **POS**.

We need n+1 distinct plans to define a n dimensional hyperplane in the n+1 cost coordinate space. We try to frame a n dimensional hyperplane in cost coordinate

space, through which the *EQUILINE* passes. If such a plane can be framed by a subset of plans optimal at the corner points we proceed to Descend_Hull otherwise we extend the partial convex hull in the *prospective direction*, by searching for an interesting plan on the *prospective face* f_{pros}.

The n+1 plans and the neighbor of these n+1 plans p, form a polyhedron(convex) in n+1 dimensions, bounded by n dimensional faces. Similarly, the *EQUILINE* will either intersect the polyhedron at 2 points(convexity of polyhedron) or does not intersect it at all. Hence the descend hull would always yield a face(n dimensional plane) having lower B(p) value out of all the vertices of the polyhedron.

3.2.1 Computing the best affine approximation

In step 1 we optimize at the corner points of the parameter space. Let pmset be the multiset of plans p_1,p_2,p_3,...,p_k, k = 2^n optimal at the corners and let $\mathbf{p_{distinct}}$ be the set of distinct plans among these. We check the cardinality of the $\mathbf{p_{distinct}}$ set.

If the cardinality of the set $\mathbf{p_{distinct}}$ is one, then we return the only plan as in four parameters case. And if the cardinality of set $\mathbf{p_{distinct}}$ is less than n+1 we search for some more plans along the *prospective direction*, that is on the prospective face f_{pros} of the parameter space. A prospective face is the bounding face of the parameter space, on which the prospects of finding a plan of our interest are better.

The *EQUILINE* will intersect the n+1 dimensional polyhedron either at two points or does not intersect at all. The boundary conditions can also be suitably handled. If the *EQUILINE* ideally intersects the polyhedron in two faces, the face having the minimum B(p) plan, that is B^{-1}(minimum(B(p_1),B(p_2),B(p_3),...,B(p_{n+1}))) is selected.

If the polyhedron is not intersected by the *EQUILINE* we search for a possibility by developing the *partial Hull along the prospective direction* by searching on prospective face in parameter space. If *getintplan* returns a plan which does not form an n dimensional intersecting plane it implies that whole of convex hull is not intersected by the *EQUILINE*. Since the boundary of the convex hull represents the parametric optimal plans along the boundary of the parameter space we search the boundary.

If we could construct a (n+1) tuple of n+1 distinct plans through which the *EQUILINE* passes we descend to the base of the convex hull using the *Descend_Hull* algorithm.

3.3 Computational Analysis

The total computational overhead can be analyzed as below. In the whole approach, the time required for the *optimize* function is at premium and must be reduced. In step 1 of *main algorithm*, optimizing at the corners of the parameter space, in the worst case makes 2^n calls of *optimize function* for n dimensional parameter space and n+1 parameter metric.

The *Descend_Hull* algorithm is the other function which also make calls to optimize function. The number of calls to optimize function is same as the number of iterations of the Descend_Hull loop.

All other procedures are basically mathematical computations done on convex hull in the cost coordinate space. The *getprospective* declares any face f_i of the parameter space as prospective only if each n+1 subset of the plans optimal at the corners of the face f_i, the nearest plane is the real one. It doesn't consider further processing for a face f_j once one of its subset has non real plane as nearest one. This also gives us lot of savings.

The *decision that EQUILINE is not intersecting the hull* too gives us savings by returning some plan on the base of the hull for this case. By optimizing in the neighborhood of the base plan we can get the least B(p) plan, as base $\subset$ Boundary.

Thus the above approach has good performance.

Chapter 4

Conclusions and Future Work

This section concludes by summarizing the work done in this thesis. We also suggest some future directions for research in the area of parametric query optimization.

4.1 Conclusions and Summary

Algorithms have been developed for solving the query optimization problem when the query metrics are non parametric in nature. These complex query metrics occur in parallel and distributed query processing environments. The very nature of query metric does not allow us to apply dynamic programming approaches.

We tried to solve the query optimization problem of such non-parametric query metrics, by expressing a set of n *Affine problems*, and applying PQO techniques to it.

We basically build our approach on the result that convex hull of plans in cost coordinate space represents the Parametric optimal set of plan [Ganguly98]. We proved that the boundary of the convex hull represents the parametric optimal set of plans along the boundary of the parameter space.

The algorithm has been developed for the cost metric

$$B(p) = max(a_0(p), a_1(p), \ldots, a_n(p))$$

The algorithm has been generalized to work for n parameter cost metrics. The above approaches can be used for other monotonous query metric like

$$B(p) = \Phi(a_0(p), a_1(p), \ldots, a_n(p))$$

Every monotonous function decreases along the convex hull from the boundary of the hull towards the base of the hull. The EQUILINE in the above case has the unique property that its intersection with the convex hull in cost coordinate space gives the best approximate of minimum B(p) plan. The descend to the base of the hull is done in the parameter space by descend_hull algorithm.

For any monotonous cost metric, we just need to find such a curve like EQUILINE, *DESCLINE* to descend to the base of the hull.

4.2 Directions for Future work

Parametric Query Optimization is relatively an unexplored area having large scope of problems to be solved. In this thesis we tried to apply PQO techniques to one type of non-parametric query metric like $B(p) = max(a_0(p), a_1(p), \ldots, a_n(p))$. Lots of issues still remain untouched.

- One immediate avenue is to apply PQO to solve query metrics like

  $$B(p) = a_0^k(p), a_1^k(p), \ldots, a_n^k(p))$$

  which occur in parallel query processing environments.

- Developing some approaches to solve for non strictly monotonous query metrics is also a nice theoretical work.

- Looking the above and other existing algorithms from implementation point of view is a nice work.

- Developing a parametric query optimizer for distributed databases and parallel databases is also a good work.

Bibliography

[Anjali99] Anjali V. Betawadker "Query Optimization with One Parameter", Sumit Ganguly(Thesis Supervisor), Indian Institute of Technology, Kanpur Technical Report MT-CS-99-04

[Ant93] G. Antoshenkov, "Dynamic Query Optimization in Rdb/VMS",*Proceedings of the IEEE International Conference on Data Engineering*, Vienna, Austria, April 1993.

[CG94] R. Cole and G. Graefe, "Optimization of Dynamic Query Evaluation Plans", *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, Minneapolis, Minnesota, USA.

[Ganguly98] S. Ganguly, "Design and Analysis of Parametric Query Optimization Algorithms", *Proceedings of the 1998 International Conference on Very Large Databases, New York, USA.*

[GHK92] S. Ganguly, W. Hasan and R. Krishnamurthy, "Query Optimization for Parallel Executions", *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data.*

[GK94] S. Ganguly and R. Krishnamurthy, "Parametric Query Optimization for Distributed Databases based on load conditions", *Proceedings of the 1994 International Conference on Management of Data, Pune, India*

[INS+92] Y.E. Ioannidis, R.T. Ng, K. Shim and T.K. Sellis,"Parametric Query Processing", *Proceedings of the 1992 International Conference on Very Large Databases*, Vancouver, British Columbia, Canada.

[KS86] Henry F. Korth and Abraham Silberschatz. *Database System Concepts.* McGraw-hill Inc, USA, 1986.

[NAEL94] Shamkant B. Navathe and Ramez Elmasri *Fundamentals of Database Systems.* ADDISON-WESLEY, 1994.

[SAC+79] P.G. Selinger, M.M. Astrahan, D.D. Chamberlain, R.A. Lorie and T.G. Price, "Access Path Selection in a Relational Database Management System", *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data.*

[SGPrUmAn2001] S. Ganguly, V.G.V. Prasad, S.V.U.M. Rao, A. Betawadkar. A Framework for parametric query optimization.

[SVUMRao99] S.V.U.M. Rao "Parametric Query Optimization: A Non-Geometric Approach", Sumit Ganguly (Thesis Supervisor), Indian Institute of Technology, Kanpur Technical Report MT-CS-99-20

[Prasad99] Varakur Ganga Vara Prasad "Parametric Query Optimization: A Geometric Approach", Sumit Ganguly (Thesis Supervisor), Indian Institute of Technology, Kanpur Technical Report MT-CS-99-24